A Home Control System Based on Fuzzy Logic

WALTER BANKS, ASHOK PATEL & SHERIF ABDEL-KADER

After reminding us of the basics of fuzzy logic, Walter and his associates concentrate on the needs of environmental control. They specifically look at some of the implementation quirks that come with writing fuzzy logic code for a home-control system.

This project started as an exercise in software portability and fuzzy logic. After working on microcontroller projects for years, we wanted to look at some of the issues in porting software between execution platforms.

We chose a fuzzy-logic-based home environment control system because it's an application that is well understood and offers many opportunities for experimentation. As well, we can put a fuzzy-logic tool that we've been using through the paces. The tool extends C to linguistic variables as well as conventional 8- and 16-bit integers.

FUZZY LOGIC

Fuzzy logic adds the concept of linguistic variables to the variable types most people are familiar with.

Linguistic variables describe data in application-specific terms. We can say, for example, that the day is hot. Further, we can qualify the linguistic term by scaling it between fuzzy 0 and fuzzy 1 (usually represented by 0 and 255). A hot day might have a crisp value of 100°F and a linguistic value of fuzzy 1. At 60°F, it is barely hot (perhaps fuzzy 0.2). At 40°F or so, it is not hot at all.

Our ability to naturally describe our problems in linguistic terms makes fuzzy logic easy to use. A fuzzy rule might be:

```
IF room IS hot THEN
  ac is on_high
```

This rule refers to two linguistic variables hot and on_high. It is interesting that linguistic variables are context sensitive.

It has been a few years since interest in fuzzy-logic technology for embedded applications started. And, it is surprising that computing has taken so long to accept fuzzy-logic technology. For the most part, fuzzy logic presents a cleaner, clearer interface to the real world—a way of expressing ourselves in terms of technology's application, and not in terms of the fundamental science on which the technology is based.

One problem with finding suitable applications comes from attempts to implement applications in fuzzy logic based on the underlying science of an application. We were competing with implementations of the science describing a real-world application, rather than two implementations of an actual application.

This small difference is significant. Most scientific application models are linear representations of the application. Implementations with fuzzy logic at best equal alternative implementations. On the other hand, many contain some nonlinear aspects, which fuzzy logic easily handles.

For example, consider a simple thermostat control system for a living room. To control the temperature, we start with two decisions: "when the room is cold, turn on

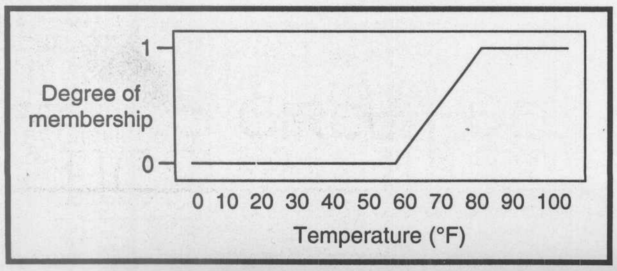


Figure 1: *This graph compares room temperature with degree of membership to determine the linguistic variable* HOT.

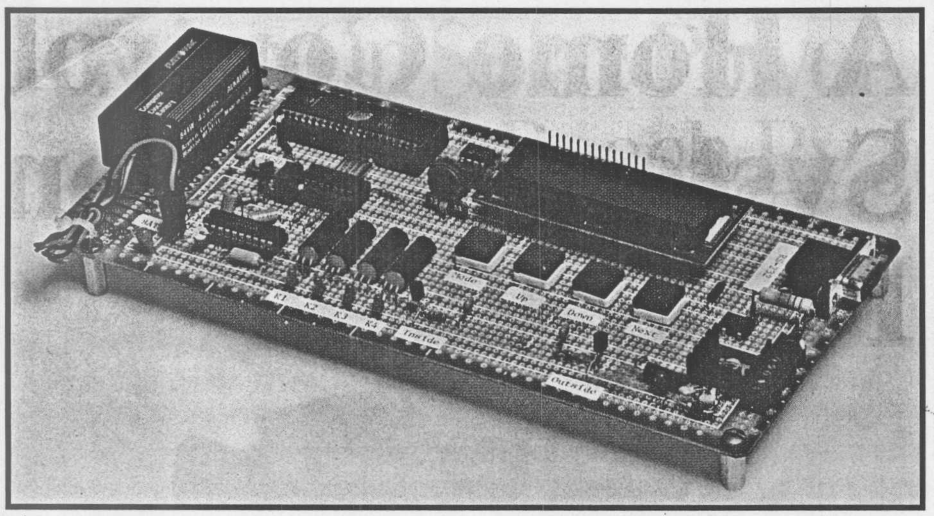


Photo 1: *There is nothing critical about either component layout or wiring on this fuzzy-logic thermostat prototype board. Clock speeds are low and the analog signals (0–5 V) are significantly above the noise levels on the board.*

the heat" or "when the room is too hot, turn off the heat."

We express these ideas formally to the computer with the following rules:

```
IF room IS Cold THEN heat IS on
IF room IS Hot THEN heat IS off
```

Now, suppose we move to Arizona, where there are cold and hot temperatures. To cope with the heat, we add rules to our system:

```
IF the room IS Cold THEN
  airConditioner IS off;
IF the room IS Hot THEN
  airConditioner IS on;
```

The control system is implemented so that it gathers a collection of independent thoughts about a problem presuming that each idea is part of the total solution.

This all sounds good, but will it work? We've introduced the concept of linguistic variables that have some meaning in the real world. Let's look at the linguistic variable, "`room IS Hot`."

`room` really indicates room temperature, so let's build a scale of temperature on the *x*-axis. On the *y*-axis, we normalize our conclusions between 0 and 1. (Boolean logic has only two values 0 and 1, while fuzzy logic has all the analog values of 0–1.)

We begin with 100°F, which we all agree is hot. We set that value to 1. We also agree that 90°F is still hot, so its value is 1 as well. At 70°F, we begin to have differing opinions. Some people say 70°F is not really hot, while others still insist it is. To account for differing opinions, we set the value at 70° to 0.5. Finally, when the temperature drops to 60°F, we all agree it's no longer hot. We set the value there to 0. Figure 1 graphically depicts this change in membership.

Now that we've defined the linguistic term `HOT` for our computer, it's simple to write a subroutine. On a scale between 0 and 1, the code needs to return how relevant `HOT` is given a temperature value. For more granularity, increase the range of possible return values from, for example, 0 to 255.

The code only requires a few instructions to implement on most computers, even the simple ones used in household appliances. `HOT`, in this example, describes a condition in the room (i.e., it is a member of room).

The code fragment in Listing 1 passed through Fuzz-C, a software preprocessor. The linguistic variable `HOT` is defined as a trapezoid (i.e., it has four arguments). The arguments are formed from the crisp intersection for each corner of the trapezoid.

The Fuzz-C preprocessor generates C code from this definition (see Listing 2). As you can see, it does two things. First, it declares `room` as type `char` (its crisp value), and second, it generates a function which takes the room temperature as an argument and returns a value between fuzzy 0 and fuzzy 1, indicating the relevance of linguistic variable `HOT`.

Even though Listing 2 shows just one linguistic variable, in this application, the crisp variable `room`

Listing 1: *The linguistic variable HOT is defined as a having a trapezoidal-shaped membership function. The definition corresponds to the graph.*

```
LINGUISTIC room TYPE char MIN 0 MAX 150
{
  MEMBER hot { 60,  90,  150, 150 }
}
```

Listing 2: *This code translates the linguistic variable from a definition to an executable function. The function can be easily implemented on most small embedded microcontrollers.*

```
char  room;

char room_hot (char __CRISP) {
  if (__CRISP < 60)
    return(0);
  else {
    if (__CRISP <= 90)
      return(((__CRISP - 60) * 8) + 7);
    else {
      return(255);
    }
  }
}
```

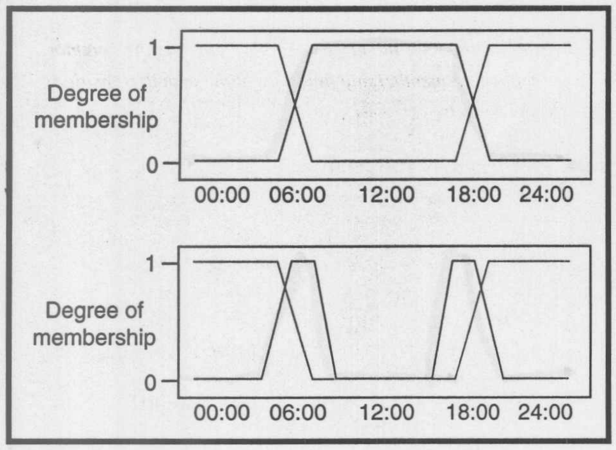


Figure 2: *The first graph compares 24-hour time and linguistic variables* `day` *(green) and* `night` *(red). The lower graph compares 24-hour time and linguistic variables* `morning`, `evening`, *and* `night setback` *(red).*

may have many linguistic members: `HOT`, `COLD`, and `NORMAL`.

Somewhat more interesting are the linguistic variable's declarations related to time of day. With the goal of referring to time in terms we use, divide the day into 240 parts. In a 24-h day, 6-minute resolution nicely fits into an 8-bit variable while still providing the resolution necessary to deal with control problems.

In our application, there are six linguistic variables in a 24-h period: `day`, `night`, `morning`, `evening`, `daysb` (day setback), and `nightsb` (night setback). It is easy to add more linguistic variables or to change the way they are defined or used.

`day` is defined as a trapezoid starting at 5:30 A.M. By 6:30 A.M., it is fully day and continues to be so until 5:30 P.M. when it starts to taper off. By 6:30 P.M., it is no longer day.

`night` is defined as `NOT day`. `night` therefore is a complement of `day` and the sum of `day` and `night`'s degree of memberships is always `fuzzy_one`.

The linguistic variables `morning` and `evening` are declared with trapezoids in the same manner, except with the appropriate time periods.

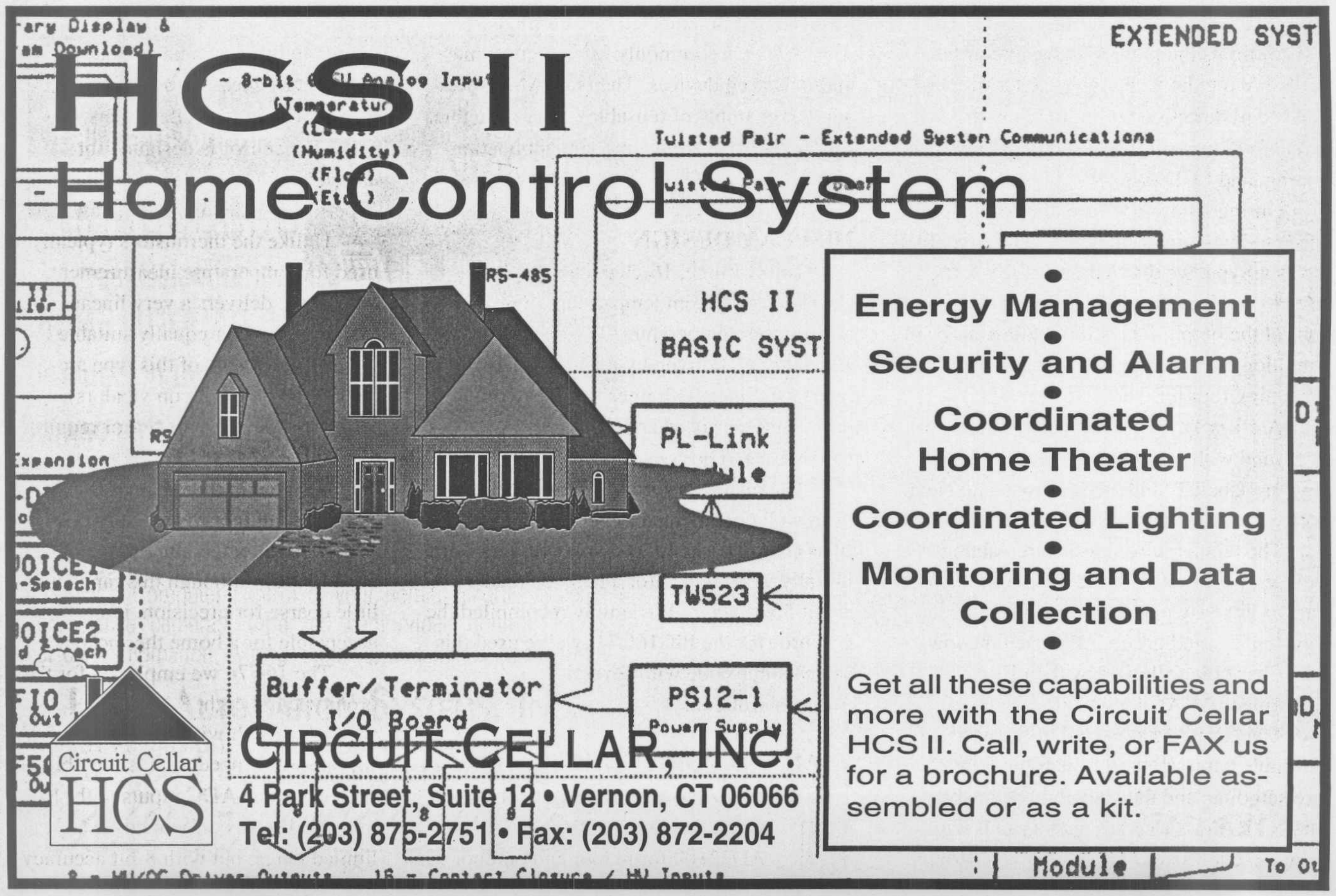
`nightsb`, the last linguistic variable, also uses a fuzzy expression. It has been declared as `night` not including `evening`. Similarly, `daysb` is also a fuzzy expression stating the day setback temperature time is `day` not including `morning` and `evening`.

Fuzz-C enables you to use variables instead of constants to declare linguistic variables. Using fuzzy expressions to generate some of the linguistic variables results in a few fixed variables which define several linguistic variables.

Generating code to produce degrees of membership is not difficult—either by hand or with some development aid. Figure 2 and Listing 3 offer a full definition of the linguistic variables associated with time on the thermostat project.

HARDWARE

The fuzzy logic thermostat prototype (see Figure 3) was implemented around a



HCS II
Home Control System

- Energy Management
- Security and Alarm
- Coordinated Home Theater
- Coordinated Lighting
- Monitoring and Data Collection

Get all these capabilities and more with the Circuit Cellar HCS II. Call, write, or FAX us for a brochure. Available assembled or as a kit.

Circuit Cellar HCS

CIRCUIT CELLAR, INC.
4 Park Street, Suite 12 • Vernon, CT 06066
Tel: (203) 875-2751 • Fax: (203) 872-2204

#202

variables associated with the crisp variable hours *define common references to the time of day. The shape of the membership functions allow smooth transitions between adjacent or overlapping time periods.*

```
LINGUISTIC hours TYPE char MIN 0 MAX 240
// hours; .1 hour 0 .. 240 for a day
{
  MEMBER day     { 55 ,  65 ,  175 , 185 }
  MEMBER night   {FUZZY { hours IS NOT day }}
  MEMBER morning {50,  60 , 80 , 90 }
  MEMBER evening { 160 ,170 ,190 , 200 }
  MEMBER nightsb {FUZZY { hours IS night AND hours IS NOT
    evening }}
  MEMBER daysb   {FUZZY { hours IS day AND hours IS NOT
    evening AND hours IS NOT morning }}
}
```

Listing 4: *This routine converts the ADC readings from the Analog Devices AD22100 sensor into actual temperature readings.*

```
#define F_const  400
#define F_offset -78*256

#define C_const  222
#define C_offset -61*256

char convert_temp (char temp) {
  return ((((long)temp * F_const) + F_offset) >> 8);
}
```

requirements of our project: several analog inputs, low-power capability, and more ROM space than we expected to need even for performance-data gathering.

The design was expected to use a 32-kHz crystal for the processor clock. While building the prototype, we decided it would be useful to add a serial port so a PC could monitor the execution of the fuzzy-logic controller. The serial port meant we needed a faster processor clock so we could keep up with the serial data port. During development, we used a 4-MHz crystal.

Household thermostats have standard wiring in most homes. The heating, air conditioning, and fan systems are controlled by contact closures to a 24-VAC supply. The humidifier relay is a contact closure brought out to two terminals. Although it appears that some form of triac-based electronic switch could be easily used, one of us had an experience where a new high-efficiency system would not operate with a triac switch. Instead, we used a relay that is compatible with a broad spectrum of heating systems.

The 24-VAC supply lines to the thermostat power the fuzzy-logic thermostat controller. The 24-VAC supply passes through a bridge rectifier, and a series resistor limits the power dispassion of the 7805 5-V regulator. We provided a simple power-fail detection circuit to allow the software to turn off the heating, air conditioning, and LCD display.

During a power failure, the software needs to maintain only the date and time. In the prototype, we divided one of the ports into two 4-bit portions. Four of the bits control the output relays for heating, air conditioning, fan, and humidifier. The remaining four bits sense key presses.

A 4-key keypad makes contact closure to ground with 10-k pull-up resistors. The keys are labeled Mode, Up, Down, and Next. Software filters the key bounce.

The temperature sensors are Analog Devices AD22100s. These three terminal sensors are easy to use—just connect between V_{CC} and ground. We used the low-pass filter (1 k followed by 0.1 µF) recommend by Analog Devices.

The NVRAM stores all of the user-definable parameters such as room temperature setpoints and day and night setbacks. The NVRAM is a serial memory part with data and address information passed serially so it can be implemented in an 8-pin part. This device is commonly used in consumer and industrial devices. The NVRAM code is another example of reusable driver code that originally came from a modem application using the Zilog Z8.

DISPLAY DESIGN

The two-line, 16-character display shows current room temperature, humidity, and outdoor temperature. The display is a standard LCD driven by a 4- or 8-line interface. Since both interfaces are available, only interface speed and available processor I/O lines need be considered.

The display's software interface has been well documented in various publications (*INK* 8). The LCD display drivers were initially written in C for a barcode reader about 5 years ago. We simply recompiled the C source for the PIC16C74 (we've used this same source code with several microcontrollers).

TEMPERATURE CONVERSION

As we mentioned before, we chose the Analog Devices AD22100 for indoor and outdoor temperature sensing. Three-terminal temperature sensors have made temperature measurements by microcontroller-based systems very easy. The sensor is designed for automotive applications that normally experience wide temperature swings.

Unlike the thermistors typically used for temperature measurement, this sensor delivers a very linear 22.5 mV per C° (other equally suitable solid-state sensors of this type are offered by other silicon vendors).

This temperature sensor requires V_{CC} and ground and produces an analog signal in the range of 0–5 V. The working range is actually 0.25–4.75 V for a temperature range of –50–+150°C. Although this range is a little coarse for precision, it's acceptable for a home thermostat.

The 16C74 we employed for the prototype has eight 8-bit A/D converters each with a range of 0-5 V. If precision is needed, it is possible to implement the ADC inputs with expanded scales only to cover a limited range, but with 8-bit accuracy.

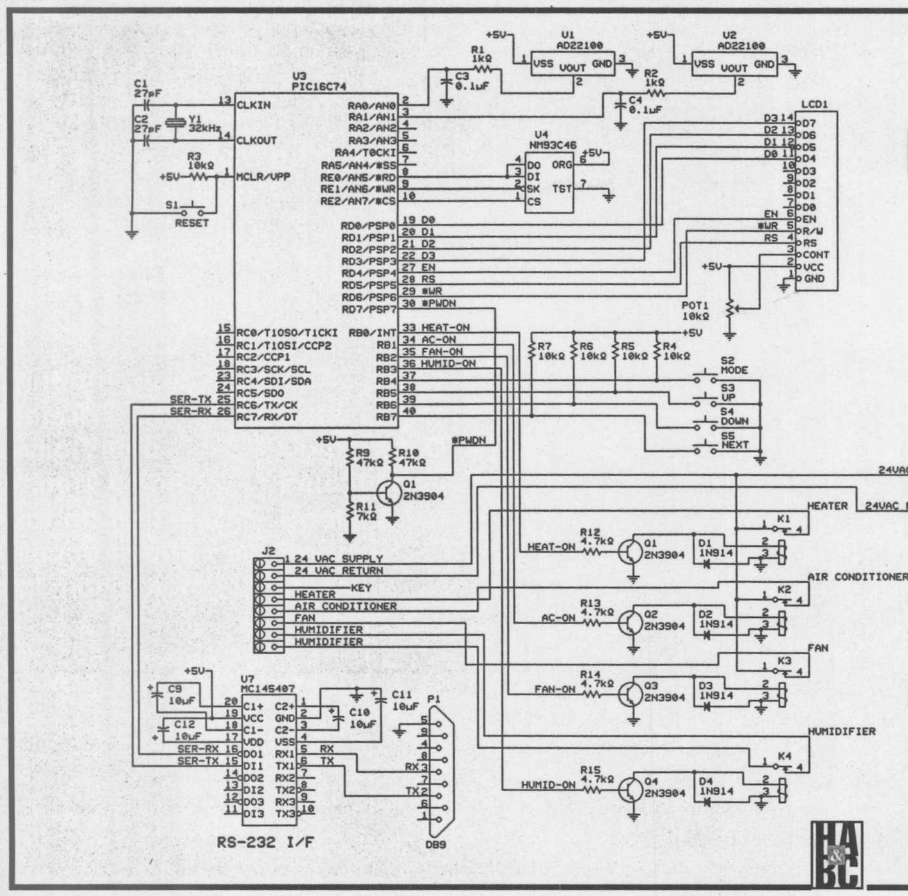


Figure 3: *The fuzzy HVAC controller is based on a Microchip PIC16C74 microcontroller. The unit calls for heating, air conditioning, fan, and humidity using simple relays. The user interface is through a small LCD display and push buttons.*

HOME AUTOMATION

Call 24hrs for FREE
64 page Color Catalog

Largest Selection of HOME AUTOMATION products in the World!

Browse us on the WEB!
http://www.techmall.com/smarthome
Over 500 unique products!

Hundreds of hard-to-find automation, X10, and wireless control products. Computer control of your home, security systems, surveillance cameras, infra red audio/video control, HVAC, pet care automation, wiring supplies, and much, much more!

HOME AUTOMATION SYSTEMS, INC.
151 Kalmus Drive, Ste L4, Costa Mesa, CA 92626
Questions 714-708-0610 FAX 714-708-0614
Call 800-SMART-HOME
800-762-7846

#203

Microcontroller Networks (μLANs)

With Cimetrics' 9-Bit μLAN you can link together up to 250 of the most popular 8- and 16- bit microcontrollers (8051, 80C196, 80C186EB/EC, 68HC11, 68HC16, PIC16C74).

The 9-Bit μLAN is:

- ***Fast***— A high speed (62.5k baud) multidrop master/slave RS-485 network
- ***Flexible***— compatible with your microcontrollers
- ***Reliable***— robust 16-bit CRC and sequence number error checking
- ***Efficient***— low microcontroller resource requirements (uses your chip's built-in serial port)
- ***Friendly***— Simple to use C and assembly language software libraries, with demonstration programs
- ***Complete***— includes network software, network monitor and RS-485 hardware
- ***Practical***— applications include data acquisition and distributed control

Microcontroller or PC Master

Intel 8051 HVAC/R Lighting Control

Intel 80C186EB/EC Point-of-Sale

Motorola 68HC11 Process Control Transportation

Microchip PIC16C74 Fire Detection Security

9-Bit RS-485 Multidrop Network

Cimetrics TECHNOLOGY

Cimetrics Technology • 55 Temple Place • Boston, MA 02111-1300 • Ph 617.350.7550 • Fx 617.350.7552

#204

to actual temperature by multiplying by a constant and adding a zero-count offset. Since we know the environment's working range, we don't need range checks on the data. In Listing 4, the code converts the ADC result to Fahrenheit. Constants for Celsius are also given.

PUTTING IT ALL TOGETHER

We now have all the pieces we need—displays, keypad, temperature sensors, reed-relay outputs, and definitions for time and comfort. Although we define our comfort in absolute temperature terms, it is easier to use a conventional thermostat setpoint to set the desired temperature and let the linguistic functions of comfort (`COLD`, `NORMAL`, and `HOT`) be based on the relative temperature error. Figure 4 shows the temperature comfort of the room. A value of 0 means the room's temperature is at the setpoint.

As we established earlier, fuzzy rules are a series of individual statements about the problem. Listing 5's partial source of the fuzzy-control block shows some of the rules used in this project.

The first few rules are obvious. We then move to the energy-saving rules. Here, given information we have about inside and outside temperatures, we make choices based on natural warming or cooling.

With multiple rules, we have to decide how we want the computer to evaluate an outcome. Typically, it would evaluate each rule independently and weigh each rule by the strength of its logical argument.

Another method involves *consequence functions*. These functions are fuzzy logic's way of resolving conflicts in the decision-making process and provide what is referred to as *defuzzification*. Fuzzy literature details many ways of doing this, each author extolling the virtues of a preferred method.

We used one of the standard methods called *center of gravity* for our consequence functions. In general, this method sums the results and the weight (degree of membership) of each input, and computes a single answer to accurately reflect all of the results. In effect, center of gravity provides smooth transitions between competing terms.

Listing 5: *This fuzzy control function regulates the heating and air conditioning from fuzzy rules. Fuzzy rules are independent intuitive control functions. The consequence functions resolve the actual control settings for heat and air conditioning.*

```
FUZZY room_control {
  IF room IS cold THEN
    ac   IS OFF
    heat IS ON

  IF room IS normal THEN
    ac   IS OFF
    heat IS OFF

  IF room IS hot THEN
    ac   IS ON
    heat IS OFF

  IF room IS cold AND OutsideTemp IS hot THEN
    heat IS OFF

  IF room IS hot AND OutsideTemp IS cold THEN
    ac IS OFF
}
```

This project has simple on and off actions even though the control calculations are done for continuous control. Consequence functions still are able to resolve different control schemes with varied inputs (see Listing 6).

All of this is tied together with a C `main` function shown in Listing 7. This function is conventional controller code which reads the indoor and outdoor temperature, calls the fuzzy-control function, then calls the LCD information display routine. The day and night setback routines are an unusual use of a linguistic variable that gives a smooth temperature transition throughout the day. The define for `NightSetback` shows how linguistic functions may be called individually.

PROTOTYPE

We built the prototype system on a small development board (see Photo 1). The photo shows the noncritical nature of the construction of the project. All of the analog signals are noncritical in the 0–5-V range. Layout of the digital signals is noncritical due to the low clock speeds used in this project.

We made a number of small changes to the design to accommodate the developmental nature of the project. First, we added a serial port to the processor so we could monitor the internal state of the thermostat. To achieve acceptable serial data rates, we increased the processor's clock speed from 32 kHz to 4.00 MHz. While this has little effect on the prototype, it increases battery drain when conventional power fails.

In hindsight, the serial port could be useful. As well as fulfilling a monitoring function, it could set the thermostat's operating parameters.

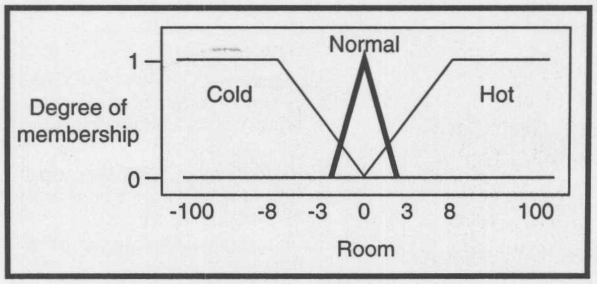


Figure 4: *You can combine linguistic terms to provide for room variables as a function of temperature errors: normal (bold), cold, and hot.*

PROJECT PORTABILITY

Within the last few years, most manufacturers have introduced versions of their microcontrollers—Microchip PIC16C74 (what we used), Zilog Z8, National COP8, and Motorola MC68HC05 and MC68-HC08—which meet this project's requirements. We didn't discount the 8051 families, but were more familiar with software tools for the rest of the chips mentioned.

We recompiled the source code for the project on each of the above microcontroller platforms. In each case, the necessary changes to the source code were limited to replacing the processor descriptor header files and making minor changes to port and bit assignments.

The most significant change is the A/D conversion method used on each of the processors. Most of the embedded 8-bit processors have at least one family member with onboard A/D conversion capability. Realistically, you could produce a demonstration board with several processor sockets—one for each supported processor.

FUZZ-C

We used the Fuzz-C preprocessor to add linguistic variables to our C program. This inexpensive preprocessor works with almost any C compiler. Fuzz-C is actually implemented as a simple compiler: it reads in a mixture of C and fuzzy-logic declarations and functions and reproduces the C unchanged. The fuzzy logic is translated into its C equivalent.

Like most fuzzy-logic projects, this one contains a little fuzzy-logic-related code (linguistic variables, consequence functions, and fuzzy-logic control functions). It also has a lot of traditional code required to scan keyboards, drive displays, control relays, and send and receive serial characters.

Fuzzy logic is starting to be seen as a tool that can add significantly to a developer's ability to describe a problem. Fuzzy-logic techniques enable developers to deal with applications that are not well defined in the crisp sense or have many different operating parameters, some of which may be nonlinear.

The files posted on Circuit Cellar BBS contain both the source code for the project and the intermediate C code and its compiled form. You can easily retarget the intermediate C code for other platforms.

FUTURE WORK

It's reasonable to tie the thermostat to a central home controller bus such as that used by the HCS II. You could then centrally control and monitor an HVAC system and room environments.

All engineering projects fall victim to three fundamental shortages: lack of information, time, and resources. Of the three, the last is the easiest to solve and the first, impossible.

Listing 6: *Here's the consequence function for heat. The defuzzification method selected is center of gravity, which means each call to the consequence function from the fuzzy rules is weighted. After all of the rules have been executed, the consequence function returns a fuzzy result. The action statement turns the heat on if the result is greater or equal to 128.*

```
CONSEQUENCE heat TYPE char MIN 0 MAX 255 DEFUZZ cg
ACTION { Heat_On  = heat >= 128;}
{
  MEMBER OFF      {   0 }
  MEMBER ON       { 255 }
}
```

Listing 7: *The complete mainline for the fuzzy home environment controller ends up being very simple. Most of the software in this system is ordinary C for scanning keyboards, updating time, driving displays. A little of the code is dedicated to linguistic variables and fuzzy functions.* `NightSetback` *shows another way that the linguistic variable* `nightsb` *may be used to get smooth transitions between different time periods. The fuzzy control function* `room_control()` *is called as a C subroutine.*

```
#define NightSetback() ((nightsb(hours) * NightOffset) / 256)
void main (void)
{
  Init_all();
  while(1) {
    if (keypressed)
      Service_mode;
    else {
      InsideTemp =  convert_temp(ReadAD(Inside));
      OutsideTemp = convert_temp(ReadAD(Outside));
      room = InsideTemp-Room_SP-NightSetback()-DaySetBack();
      room_control();
      Display();
    }
  }
}
```

Probing the problem illuminates the science involved, but to achieve results, we must still accomplish the engineering. Fuzzy-logic-based systems address both science and engineering with a single brilliant idea: "Why not describe the final solution to the problem intuitively?" Engineering and science then become side effects of the solution rather than the other way around.

Walter Banks is the President of Byte Craft, a company specializing in code creation tools for embedded microcontrollers. He may be reached at walter@bytecraft.com.

Ashok Patel is the president of Softart Microsystems where he does hardware and software designs using microcontrollers for the communications industry.

Sherif Abdel-Kader is a software support specialist of Byte Craft. He is a graduate from the University of Cairo has extensive experience in both embedded systems development and customer support.

SOURCE

MPC compiler for PIC16C74,
Fuzz-C
Byte Craft Limited
421 King St. N.
Waterloo, Ontario
Canada N2J 4E4
(519) 888-6911
Fax: (519) 746-6751
info@bytecraft.com

I R S

413 Very Useful
414 Moderately Useful
415 Not Useful